Министерство науки и высшего образования Российской Федерации Федеральное государственное бюджетное образовательное учреждение высшего образования

«Ульяновский государственный технический университет» Кафедра «Вычислительная техника»

Дисциплина «Системы искусственного интеллекта»

Лабораторная работа №3

«База знаний и онтологии»

Вариант № 5

Выполнил студент группы ИВТАПбд-41:

Галацков И.А.

Проверил:

Хайруллин И.Д.

Ульяновск, 2024**Постановка задачи**

В рамках данной лабораторной работы необходимо разработать систему управления для автоматизации заданного технического объекта. Для хранения базы правил и онтологии предметной области предлагается использовать PostgreSOL. Задание предполагает проектирование логической модели управления, построение и настройку правил управления, а также разработку симулятора для проверки работы системы управления. Для формирования условий срабатывания правил необходимо использовать фаззификацию на основе нечеткой логики, для формирования управляющих инструкций - дефаззификацию.

Также необходимо разработать минимально рабочий симулятор предметной области с дискретным программным управлением.

Вариант: автоматизация кормления животных на ферме: разработать систему, которая контролирует подачу корма в зависимости от времени, веса животных и температуры.

**Ход выполнения работы**

Сначала была создана база данных в PostgreSQL, которая включала таблицы, содержащие информацию о весе, периоде времени и температуре. Данные в этих таблицах, следующим образом, указываются нечеткие данные, а также конкретные значения.

Далее подключаем созданную базу данных к проекту на языке программирования Python. Далее формируем правила, для определения конкретного результата. Все правила занесены в отдельную таблицу в базе данных. Для определения правил был написан скрипт, который формирует правила, путем перебора всех возможных вариантов комбинаций входных данных и указания для них результата, после чего заносит правило в базу данных.

Далее реализуем систему применения правил и симулятор работы. Входные данных генерируются случайным образом. Далее выполняется запрос в базу данных для получения нечетких значений входных данных и после этого выполняется запрос в базу для определения подходящего правила и получения результата. После чего в консоли выводится результат работы симулятора.

**Тестирование**

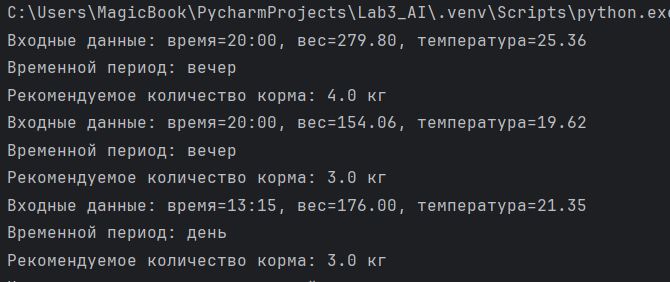
****

Рис. 1. Результат работы программы

**Вывод**

В ходе выполнения данной лабораторной работы, была создана автоматическая система контроля подачи корма для животных, исходя из веса, температуры и времени суток.

**Приложение**

import psycopg2  
from datetime import datetime  
import random  
  
   
  
#Подключение к БД  
def connect\_to\_db():  
 conn = psycopg2.connect(  
 host="localhost",  
 database="farm\_management",  
 user="postgres",  
 password="admin",  
 options="-c client\_encoding=UTF8" # Установка клиентской кодировки  
 )  
 return conn  
#Генерация входных данных  
def generate\_random\_inputs():  
 # Случайное время  
 time = random.choice(["06:30", "13:15", "20:00", "23:45"])  
 # Случайный вес животного  
 weight = random.uniform(50, 300)  
 # Случайная температура  
 temperature = random.uniform(-5, 35)  
 return time, weight, temperature  
#Определение временного интервала  
def determine\_time\_period(time):  
 conn = connect\_to\_db()  
 cursor = conn.cursor()  
 query = """  
 SELECT period\_name FROM time\_periods  
 WHERE %s::time BETWEEN start\_time AND end\_time;  
 """  
 cursor.execute(query, (time,))  
 result = cursor.fetchone()  
 conn.close()  
 return result[0] if result else None  
  
def fuzzify\_weight(weight):  
 if weight <= 100:  
 return {"легкий": 1, "средний": 0, "тяжёлый": 0}  
 elif 100 < weight <= 200:  
 return {  
 "легкий": (200 - weight) / 100,  
 "средний": (weight - 100) / 100,  
 "тяжёлый": 0  
 }  
 elif weight > 200:  
 return {"легкий": 0, "средний": (300 - weight) / 100, "тяжёлый": (weight - 200) / 100}  
#Фаззификация данных  
def fuzzify\_temperature(temp):  
 if temp <= 10:  
 return {"холодно": 1, "нормально": 0, "жарко": 0}  
 elif 10 < temp <= 25:  
 return {  
 "хородно": (25 - temp) / 15,  
 "нормально": (temp - 10) / 15,  
 "жарко": 0  
 }  
 elif temp > 25:  
 return {"холодно": 0, "нормально": (40 - temp) / 15, "жарко": (temp - 25) / 15}  
#Применение правил  
def apply\_rules(time\_period, weight\_fuzzy, temp\_fuzzy):  
 conn = connect\_to\_db()  
 cursor = conn.cursor()  
 query = """  
 SELECT fr.feed\_amount, tp.period\_name, w.weight\_category, t.temperature\_category, fr.priority  
 FROM feeding\_rules fr  
 JOIN time\_periods tp ON fr.time\_period\_id = tp.id  
 JOIN weights w ON fr.weight\_id = w.id  
 JOIN temperatures t ON fr.temperature\_id = t.id  
 WHERE tp.period\_name = %s;  
 """  
 cursor.execute(query, (time\_period,))  
 rules = cursor.fetchall()  
 conn.close()  
  
 applicable\_rules = []  
 for rule in rules:  
 weight\_degree = weight\_fuzzy.get(rule[2], 0)  
 temp\_degree = temp\_fuzzy.get(rule[3], 0)  
 degree = min(weight\_degree, temp\_degree)  
 #print(f"Категория веса: {rule[2]}, степень: {weight\_degree}")  
 #print(f"Категория температуры: {rule[3]}, степень: {temp\_degree}")  
 #print(f"Правило: {rule}, степень соответствия: {min(weight\_degree, temp\_degree)}")  
  
 if degree > 0:  
 applicable\_rules.append({"feed\_amount": rule[0], "degree": degree, "priority": rule[4]})  
  
 return applicable\_rules  
#Разрешение крнфликтов  
def resolve\_conflicts(applicable\_rules):  
 if not applicable\_rules:  
 return 0 # Нет подходящих правил  
 applicable\_rules.sort(key=lambda x: (-x["priority"], -x["degree"]))  
 best\_rule = applicable\_rules[0]  
 return best\_rule["feed\_amount"]  
#Заносим результат в БД  
def log\_results(time, weight, temp, feed\_amount):  
 conn = connect\_to\_db()  
 cursor = conn.cursor()  
 query = """  
 INSERT INTO feeding\_logs (input\_time, input\_weight, input\_temperature, feed\_amount)  
 VALUES (%s, %s, %s, %s);  
 """  
 cursor.execute(query, (time, weight, temp, feed\_amount))  
 conn.commit()  
 conn.close()  
  
#Симуляция работы  
def simulate():  
 time, weight, temp = generate\_random\_inputs()  
 time\_period = determine\_time\_period(time)  
 if not time\_period:  
 print("Не удалось определить временной период.")  
 return  
  
 weight\_fuzzy = fuzzify\_weight(weight)  
 temp\_fuzzy = fuzzify\_temperature(temp)  
  
 applicable\_rules = apply\_rules(time\_period, weight\_fuzzy, temp\_fuzzy)  
 feed\_amount = resolve\_conflicts(applicable\_rules)  
  
 print(f"Входные данные: время={time}, вес={weight:.2f}, температура={temp:.2f}")  
 print(f"Временной период: {time\_period}")  
 print(f"Рекомендуемое количество корма: {feed\_amount} кг")  
 log\_results(time, weight,temp, feed\_amount)  
  
  
  
  
  
# Функция для вставки правил в таблицу feeding\_rules  
def insert\_rules\_into\_db(rules):  
 conn = connect\_to\_db()  
 cursor = conn.cursor()  
 try:  
 for rule in rules:  
 query = """  
 INSERT INTO feeding\_rules (id, time\_period\_id, weight\_id, temperature\_id, feed\_amount, priority)  
 VALUES (%s, %s, %s, %s, %s, %s);  
 """  
 cursor.execute(query, (  
 rule['rule\_id'],  
 rule['time\_period'],  
 rule['weight'],  
 rule['temperature'],  
 rule['feed\_amount'],  
 rule['priority']  
 ))  
 conn.commit()  
 print("Все правила успешно добавлены в базу данных.")  
 except Exception as e:  
 conn.rollback()  
 print(f"Ошибка при добавлении правил: {e}")  
 finally:  
 cursor.close()  
 conn.close()  
  
# Генерация всех правил  
def generate\_unique\_rules():  
 time\_periods = [1, 2, 3] # ID временных периодов  
 weights = [1, 2, 3] # ID категорий веса  
 temperatures = [1, 2, 3] # ID категорий температуры  
 rules = []  
 rule\_id = 1  
  
 # Определение фиксированных значений корма и приоритета  
 feed\_amount\_map = {  
 (1, 1): 1.0, # Пример: light + cold → 1.0 кг  
 (1, 2): 1.5, # light + normal  
 (1, 3): 2.0, # light + hot  
 (2, 1): 2.5, # medium + cold  
 (2, 2): 3.0, # medium + normal  
 (2, 3): 3.5, # medium + hot  
 (3, 1): 3.0, # heavy + cold  
 (3, 2): 4.0, # heavy + normal  
 (3, 3): 4.5, # heavy + hot  
 }  
  
 priority\_map = {  
 (1, 1): 3, # Пример: light + cold → высокий приоритет  
 (1, 2): 2, # light + normal  
 (1, 3): 1, # light + hot  
 (2, 1): 3, # medium + cold  
 (2, 2): 2, # medium + normal  
 (2, 3): 1, # medium + hot  
 (3, 1): 3, # heavy + cold  
 (3, 2): 2, # heavy + normal  
 (3, 3): 1, # heavy + hot  
 }  
  
 for time\_period in time\_periods:  
 for weight in weights:  
 for temp in temperatures:  
 feed\_amount = feed\_amount\_map[(weight, temp)]  
 priority = priority\_map[(weight, temp)]  
 rules.append({  
 "rule\_id": rule\_id,  
 "time\_period": time\_period,  
 "weight": weight,  
 "temperature": temp,  
 "feed\_amount": feed\_amount,  
 "priority": priority  
 })  
 rule\_id += 1  
 return rules  
  
  
for i in range(5):  
 simulate()  
#rules = generate\_unique\_rules()  
#insert\_rules\_into\_db(rules)